POLITECNICO
MILANO 1863

AUDIO EFFECTS CLASSIFICATION

GRUPPO 10
MEMBERS' PERSONAL CODES:
10635718 - 10743616 - 10776190 - 10787525

Abstract

The aim of this project was to implement a classifier able to identify the audio effect used in recordings of electric guitar and bass. In particular, we developed a system able to distinguish with high accuracy monophonic sounds produced by an electric guitar when **tremolo**, **distortion** or **no effects** are applied. The same model was applied on electric bass guitar's recordings, showing good accuracy as well, though less than the latter case. Noticeably, we exploit the open source Python library *sklearn* in order to choose the best model according to the features we used.

Contents

List of Figures

List of Tables

1 Background

1.1 Database description

The IDMT-SMT-Audio-Effects database is a large database from *Fraunhofer Institute for Digital Media Technology IDMT*. The database consists of .WAV files of monophonic and polyphonic sounds. It contains recordings of 2 different electric guitars and 2 different electric bass guitars, each with two different pick-up settings and up to three different plucking styles (finger plucked - hard, finger plucked - soft, picked) [1].

Among the 11 included audio effects, we focused, as requested, on the three classes **NoFx**, **Tremolo** and **Distortion**. In order to deal with the fact that for both the electric guitar and the electric bass guitar the NoFx class was underrepresented, we used a special parameter on *sklearn.svm.SVC* to weight the three classes differently, so that we did not need to augment the database by hand. The parameter is indeed `class_weight:balanced`, the "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data [2].

1.2 Features

1.2.1 Implementation

To implement the features, most fundamental it has been the open source Python library *librosa*. For each of the following short time features, we have computed:

mean, minimum, maximum and standard deviation. Additionally, for the Mel-scaled Spectrogram, MFCC, and Spectral Contrast values we have added the delta features - local estimate of the derivative of the input data. We ended up with 364 values characterising each of the recordings[3]. See Appendix A1 for a list of the features we used.

1.2.2 Best features selection

In order to identify the most useful features for our classification problem we imported from *sklearn.feature_selection* the *SelectKBest* utility which allows to select features according to the k highest scores.

1.3 Model

1.3.1 Best model selection

We exploited *GridSearchCV* from *sklearn.model_selection* to perform **hyperparameter tuning** in order to determine the optimal values for a given model. We pass predefined values for hyperparameters to the *GridSearchCV* function. To properly use the function, we defined a dictionary in which we mention a particular hyperparameter along with the values it can take. *GridSearchCV* tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the *Cross-Validation* method. Hence, we get accuracy/loss ratio for every combination of hyperparameters and we can choose the one with the best performance [4]. We used this strategy to identify the best classifier among a collection of 5 classifiers already implemented in *sklearn* library and define, at the same time, the best hyperparameters. See Appendix A2 for a list of the models we taken into account.

2 Implementation

2.1 Data Gathering

As mentioned above, we started from monophonic electric guitar recordings. The recordings are actually divided into two folders, thus, for simplicity, we moved all the recordings of our interest (i.e., "NoFx", "Tremolo", "Distortion") in a dedicated directory. We gathered all the recordings in a list and collected the respective labels in a external .npy file. The overall number of recordings was 4368, 1872 "Distortion" recordings, 624 "NoFx" recordings, 1872 "Tremolo" recordings. This data-set is noticeably unbalanced.

2.2 Preprocessing

2.2.1 Amplitude Normalization

A plot of the minimum and maximum value in amplitude of each recording we noticed that the three classes are clearly distinguishable from their magnitude. To prevent the intensity of the sound to influence our model, we normalized the amplitude of our recordings from -1 to 1. As expected, this has shown to slight decrease the accuracy of our model.

2.2.2 Length and Sampling frequency

A quick check on the length of our recordings showed that they have been already preprocessed to have all the length of 88201 samples. They also have been recorded at the same sampling frequency of 44.1 kHz.

2.3 Features Computation

To properly compute the features we choose window parameters for windowing the signals according to the **Constant Overlapp-And-Add** condition. Thus, we selected a *Hamming* window of 1024 recordings, with hop size equal to half of the window length. We computed the features using the *librosa* library. For what concern *MFCC* and *Mel-Scaled Spectrogram*, we used a number of coefficient respectively equal to 21 and 40, as commonly reported in literature. The array of features was stored in a .npy file.

2.4 Train-Test Splitting

To randomly split the data-set in training-set and test-set we used *train_test_split* from *sklearn.model_selection*. This utility maintains unchanged the relative dimensions of each class and shuffles the recordings.

2.5 Features Analysis

2.5.1 Features Normalization

Normalization of the entire features array was done through the *minmax_scale* function from *sklearn.preprocessing* - the results were plot in figure 2.1. We used the *MinMaxScaler* object from the same package to normalize the training set and the test set, accordingly to the range defined by minimum and maximum values of our training set.

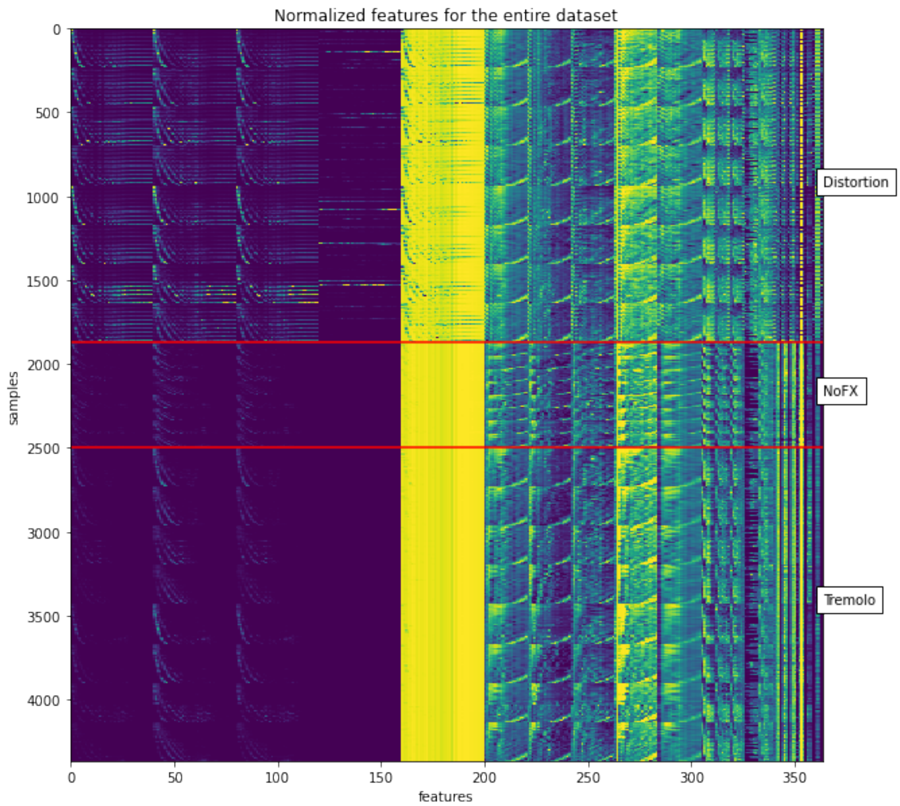

Figure 2.1: Features plot for all the samples(note: most easily distinguishable is "Distortion" class, while "NoFx" and "Tremolo" exhibit quite similar properties)

2.5.2 Best Features Selection

Using *SelectKBest* we found out the **3 most important features** to be the *standard deviation of Spectral Centroid*, the *standard deviation of Spectral Roll-off* and the *mean of Spectral Flatness*.

2.6 Model Selection and Validation

2.6.1 All-features model

Best Model Selection

Considering a model which uses the entire set of features we defined, the best model turned out to be the *SVM*. The *GridSearchCV* function allowed us to find the best hyperparamenters for our case. The worse model was instead the *KNN*.

	model	best_score	best_params
0	SVM	0.989984	{'C': 100, 'gamma': 0.05, 'kernel': 'rbf'}
1	Logistic_Regression	0.978819	{'C': 10}
2	Random_Forest	0.969951	{'max_depth': 10, 'n_estimators': 50}
3	Decision_Tree	0.950490	{'max_depth': 10}
4	KNN	0.865483	{'metric': 'minkowski', 'n_neighbors': 11}

Table 2.1: Table of scores for all-features models

Validation

Feeding our *SVM* with the test data-set we eventually got an incredibly high accuracy level, equal to 0.9965675057208238. Computing and plotting the *confusion matrix* from *sklearn.metrics* package, we could see that all the 3 classes are almost always identified.

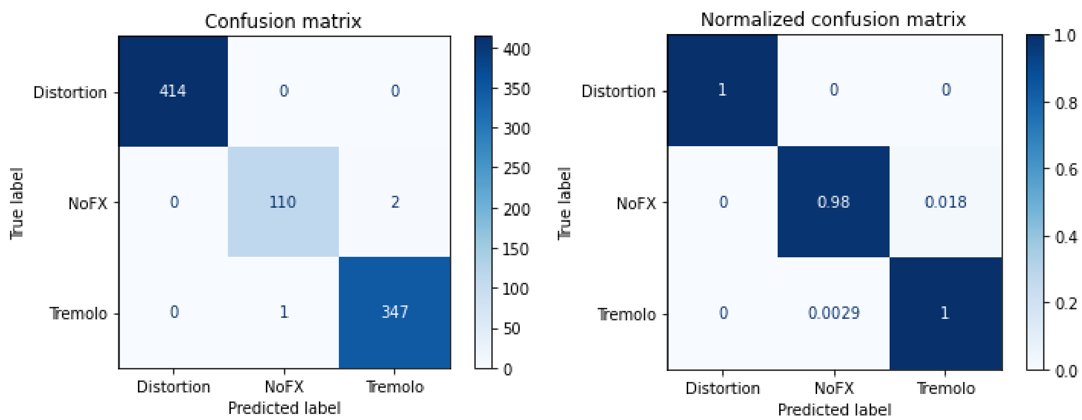

Figure 2.2: Best all-features model confusion matrix

2.6.2 Selected-features model

Best Model Selection

Using only the 3 selected features we defined earlier, this time the best model turned out to be the *KNN*, while the *Support Vector Machine* loses almost 0.1 points on score. In this case, the worse model was instead the *Decision Tree*.

	model	best_score	best_params
0	KNN	0.904981	{'metric': 'manhattan', 'n_neighbors': 31}
1	Random_Forest	0.895537	{'max_depth': 15, 'n_estimators': 100}
2	SVM	0.885801	{'C': 100, 'gamma': 1, 'kernel': 'rbf'}
3	Logistic_Regression	0.881223	{'C': 10}
4	Decision_Tree	0.856037	{'max_depth': 15}

Table 2.2: Table of scores for selected-features models

Validation

The accuracy we got using only the 3 selected features is equal to 0.9267734553775744. Our model loses near to the 7% of accuracy with the respect to the previous case. Plotting the *confusion matrix* we could see that, while "Distortion" class is still well distinguished, "NoFx" is easily confused with "Tremolo".

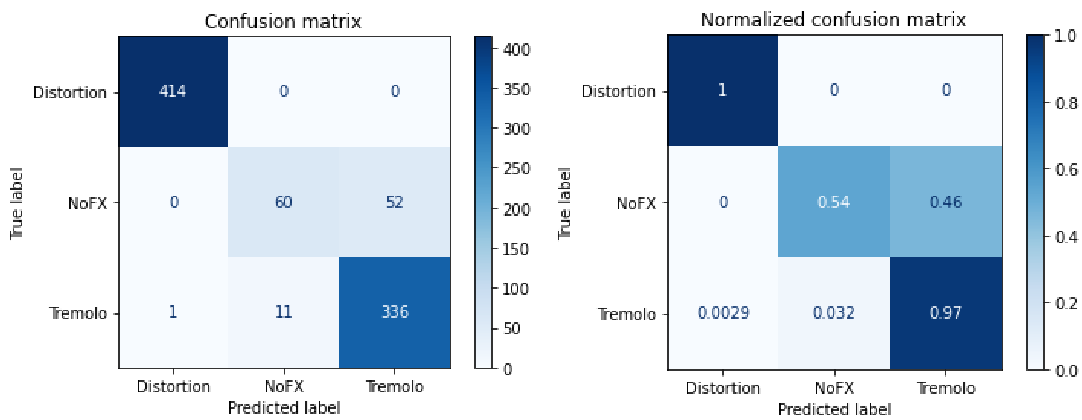

Figure 2.3: Best selected-features model confusion matrix

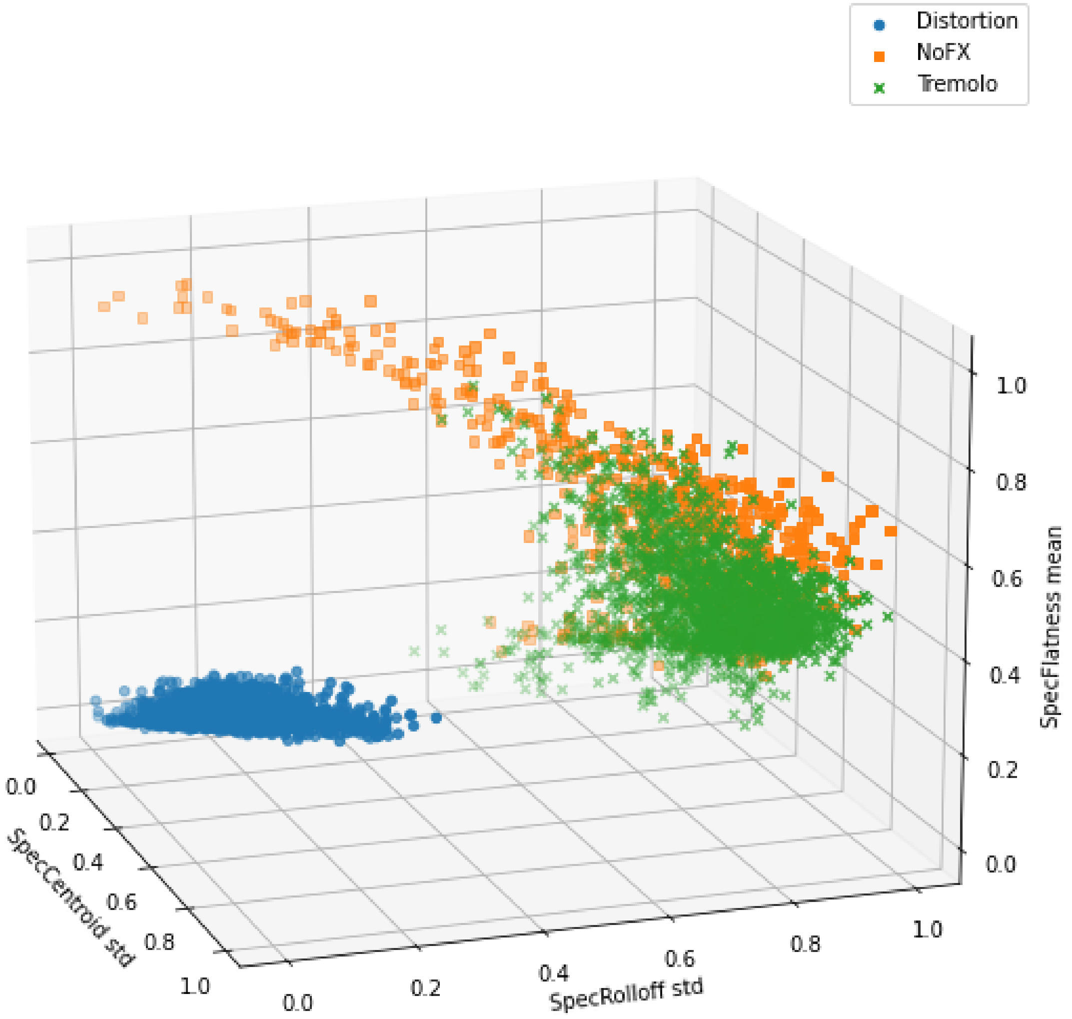

Figure 2.4: Scatter plot with the 3 k-best selected features

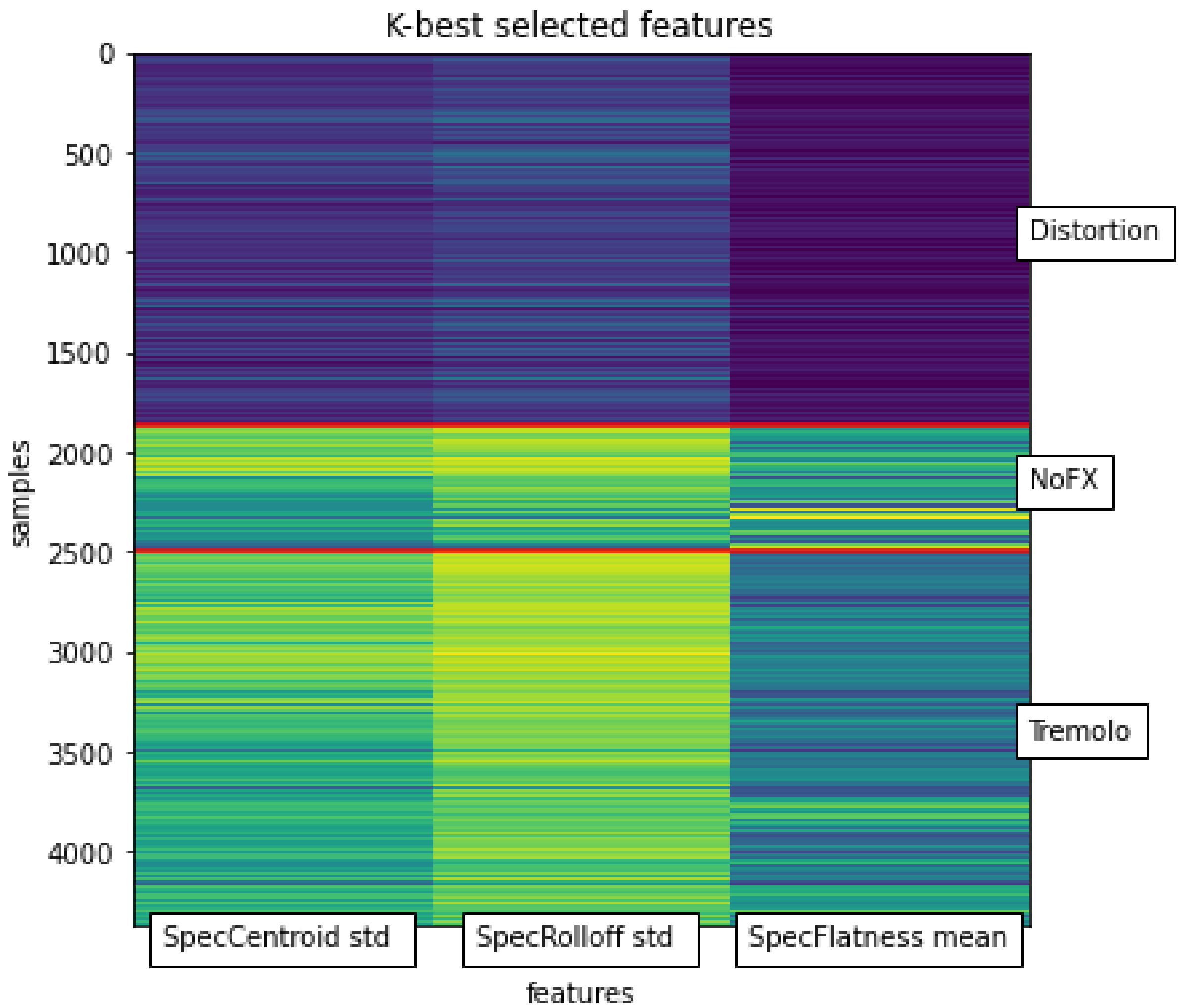

Figure 2.5: Features plot with the 3 k-best selected features

3 | Results And Discussions

3.1 Overall Conclusions On Accuracy Of Our Model

Our model is able to distinguish the 3 classes of sound with high precision. Best results are obtained in distinguishing monophonic electric guitar recordings. For what concern electric bass guitar, a slight decrease in accuracy occurs, but precision remains high. The 3 selected features are really effective in distinguishing "Distortion", but do not provide a good classification of "NoFx" sounds, which are often mistaken for "Tremolo" sounds. We didn't take into account polyphonic sounds.

3.2 Potential Improvements

Possible improvements may concern a better classification of "NoFx" sounds and "Tremolo" sounds. As can be noticed in figure 2.1, there is not a features which manages to discriminate the two classes. Further studies may lead to the definition of such a feature, and we expect it to be selected among the k best features.

Bibliography

[1] M. Stein. Idmt-smt-audio-effects. [Online]. Available: https://www.idmt.fraunhofer.de/en/business_units/m2d/smt/audio_effects.html

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] B. McFee, V. Lostanlen, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, J. Mason, D. Ellis, E. Battenberg, S. Seyfarth, R. Yamamoto, K. Choi, viktorandreevichmorozov, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, D. Hereñú, F.-R. Stöter, P. Friesch, A. Weiss, M. Vollrath, and T. Kim. (2020, Jul.) librosa/librosa: 0.8.0. [Online]. Available: https://doi.org/10.5281/zenodo.3955228

[4] H. Mujtaba. (2020) Hyperparameter tuning with gridsearchcv. [Online]. Available: https://www.mygreatlearning.com/blog/gridsearchcv/

A1 | List of features

List and brief descriptions of features we used:

- ***Spectral Centroid***: The magnitude spectrogram is treated as a distribution over frequency bins, and the centroid is extracted per frame.

$$centroid = \frac{\sum_k S(k) * freq(k)}{\sum_j S(j)}$$

  where $S()$ is the magnitude spectrum.

- ***Spectral Roll-off***: The center frequency for a spectrogram bin such that at least 85% (or another selectable percentage) of the energy of the spectrum in the frame is contained in the bins up to that one.

- ***Spectral Flatness***: Spectral flatness (or tonality coefficient) is a measure to quantify how much noise-like a sound is, as opposed to being tone-like.

- ***Zero-Crossing Rate***: Number of times that the audio waveform crosses the zero axis.

$$ZCR = \frac{1}{2} \sum_{t=1}^{T-1} |signs(t) - signs(t-1)| \frac{F_s}{N}$$

- ***Mel-scaled Spectrogram***: Compute a mel-scaled spectrogram. The mel scale is a logarithmic scale which allows to represent frequencies in a way close to the perception of pitch by a human listener, according to this formula:

$$Pitch(mel) = 1127.0148 * log(1 + \frac{f}{700})$$

- ***Mel-Frequency Cepstral Coefficients***: Those coefficients are computed starting from a series of triangular filter equally spaced on a mel-scale just discussed. At the output of each filter is applied the logarithm, and a Discrete Cosine Transform (DCT).
- ***Spectral Contrast***: Each frame of a spectrogram is divided into sub-bands. For each sub-band, the energy contrast is estimated by comparing the mean energy in the top quantile (peak energy) to that of the bottom quantile (valley energy). High contrast values generally correspond to clear, narrow-band signals, while low contrast values correspond to broad-band noise.
- ***Root-Mean-Square***: Compute root-mean-square (RMS) value for each frame.

A2 | List of models

List and brief descriptions of models we compared.

- ***Support Vector Machines***: An SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.
- ***Random Forest***: Random forests operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.
- ***Logistic Regression***: Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. In regression analysis, logistic regression is estimating the parameters of a logistic model. Outputs with more than two values are modeled by multinomial logistic regression.
- ***K-Nearest Neighbors***: In KNN an object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.
- ***Decision Tree***: In machine learning a decision tree is a predictive model where each internal node represents a variable while leaves contains the value for a certain variable.